<https://youtu.be/xZ_vg2sP4bA>

**CSS in JS**

Before we jump into how CSS in JavaScript works, let's check out an example of some "normal" HTML and CSS:

*<!-- index.css -->*

.avatar {

border-radius: 5px;

margin: 10px;

width: 48px;

height: 48px;

}

*<!-- // index.html -->*

<div>

<img class='avatar' src='https://tylermcginnis.com/tylermcginnis\_glasses-300.png' />

</div>

Nothing too surprising! But since we're not using HTML or CSS files to build mobile apps -- how would this look in React Native?

First, it important to know that all of the core components in React Native can accept a prop named style. One way we can leverage this prop is to provide styling to components with inline JavaScript objects:

**function** **Avatar** ({ src }) {

**return** (

<View>

<Image

style={{borderRadius: 5, margin: 10, width: 48, height: 48}}

source={{uri: 'https://tylermcginnis.com/tylermcginnis\_glasses-300.png'}}

/>

</View>

);

}

In the example above, the <Image> component receives two props: style and source. The value of style is just a plain old JavaScript object with borderRadius, margin, width, and height properties. Keep in mind that unlike CSS on the web, properties are written in camelCase (i.e., borderRadius in CSS in JS, but border-radius on the web).

This works, but things can get muddy quickly. Imagine if the inline object contained a dozen properties, or if we wanted the same style to apply to more than just one component! One way to keep your code DRY and reusable is to store the object in a variable:

**const** styles = {

image: {

borderRadius: 5,

margin: 10,

width: 48,

height: 48

}

};

**function** **Avatar** ({ src }) {

**return** (

<View>

<Image

style={styles.image}

source={{uri: 'https://tylermcginnis.com/tylermcginnis\_glasses-300.png'}}

/>

</View>

);

}

It's a great way to clean things up a bit, but React Native goes a step further with its StyleSheet API. Check out the following example:

**import** React **from** 'react';

**import** { StyleSheet, Text, View } **from** 'react-native';

**export** **default** **class** **TextExample** **extends** **React**.**Component** {

render() {

**return** (

<View>

<Text style={styles.greenLarge}>This is large green text!</Text>

<Text style={styles.red}>This is smaller red text!</Text>

</View>

);

}

}

**const** styles = StyleSheet.create({

greenLarge: {

color: 'green',

fontWeight: 'bold',

fontSize: 40

},

red: {

color: 'red',

padding: 30

},

});

Here, an object containing styles is passed into StyleSheet's create method. It looks similar to styling with a JavaScript object variable! However, using StyleSheet gives us a few benefits in terms of code quality and performance. We’ll take a closer look later in this Lesson as well, but this is how the React Native docs describe it:

Code quality

* By moving styles away from the render function, you're making the code easier to understand.
* Naming the styles is a good way to add meaning to the low-level components in the render function.

Performance

* Making a stylesheet from a style object makes it possible to refer to it by ID instead of creating a new style object every time.
* It also allows to send the style only once through the bridge. All subsequent uses are going to refer to an id (not implemented yet).

Another benefit is that StyleSheet validates the content within the style object as well. This means that should there be any errors in any properties or values in your style objects, the console will throw an error during compilation instead of at runtime.

## 💡 Additional Styling💡

If you wanted to implement more than one style to a component, the style prop can accept styles as an array:

`js

This will be red, then greenLarge `

The above <Text> component will render large green text, as the last style in the array will take precedence. This is a great way to inherit styles!

<https://youtu.be/VpZC-LrZwW4>

## 💡 Libraries for CSS in JS💡

Styling in React is going through a renaissance period right now just as Flux did a few years ago (which eventually gave us Redux). There are many different styling libraries popping up and each has different tradeoffs. Two of the most popular are [**Glamorous**](https://github.com/robinpowered/glamorous-native) and [**Styled Components**](https://github.com/styled-components/styled-components). The whole idea behind both of these libraries is that styling is a primary concern of the component, and therefore styling should be coupled with the component itself. We'll take a look at using Styled Component with React Native a little bit later.

True about styling in React Native:-

All of the core components accept a style property.

Style names and values always match how they are used on the web (e.g., we need to use background-color in React Native)

The StyleSheet API allows us to define multiple styles in a single place

With React Native, applications are styled using JavaScript.